2013

Hybrid Automated Reasoning Tools: from Black-box to Clear-box Integration

Marcello Balduccini
Eastman Kodak

Yuliya Lierler
University of Nebraska at Omaha, ylierler@unomaha.edu

Hybrid Automated Reasoning Tools: from Black-box to Clear-box Integration

Marcello Balduccini[1] and Yulia Lierler[2]

[1] College of Information Science and Technology
Drexel University
marcello.balduccini@gmail.com
[2] Computer Science Department
University of Nebraska at Omaha
ylierler@unomaha.edu

Abstract Recently, researchers in answer set programming and constraint programming spent significant efforts in the development of hybrid languages and solving algorithms combining the strengths of these traditionally separate fields. These efforts resulted in a new research area: constraint answer set programming (CASP). CASP languages and systems proved to be largely successful at providing efficient solutions to problems involving hybrid reasoning tasks, such as scheduling problems with elements of planning. Yet, the development of CASP systems is difficult, requiring non-trivial expertise in multiple areas. This suggests a need for a study identifying general development principles of hybrid systems. Once these principles and their implications are well understood, the development of hybrid languages and systems may become a well-established and well-understood routine process. As a step in this direction, in this paper we conduct a case study aimed at evaluating various integration schemas of CASP methods.

1 Introduction

Knowledge representation and automated reasoning are areas of Artificial Intelligence dedicated to understanding and automating various aspects of reasoning. Such traditionally separate fields of AI as answer set programming (ASP) [4], propositional satisfiability (SAT) [13], constraint (logic) programming (CSP/CLP) [22,14] are all representatives of directions of research in automated reasoning. The algorithmic techniques developed in subfields of automated reasoning are often suitable for distinct reasoning tasks. For example, answer set programming proved to be an effective tool for formalizing elaborate planning tasks whereas CSP is efficient in solving difficult scheduling problems. Nevertheless, if the task is to solve complex scheduling problems requiring elements of planning then neither ASP nor CSP alone is sufficient. In recent years, researchers attempted to address this problem by developing *hybrid* approaches that combine algorithms and systems from different AI subfields. Research in satisfiability modulo theories (SMT) [21] is a well-known example of this trend.

More recent examples include constraint answer set programming (CASP) [16], which integrates answer set programming with constraint (logic) programming. Constraint answer set programming allows to combine the best of two different automated

reasoning worlds: (1) modeling capabilities of ASP together with advances of its SAT-like technology in solving and (2) constraint processing techniques for effective reasoning over non-boolean constructs. This new area has already demonstrated promising activity, including the development of the CASP solvers ACSOLVER [19], CLINGCON [11], EZCSP [2], and IDP [25]. Related techniques have also been used in the domain of hybrid planning for robotics [23]. CASP opens new horizons for declarative programming applications. Yet the developments in this field pose a number of questions, which also apply to the automated reasoning community as a whole.

The broad attention received by the SMT and CASP paradigms, which aim to integrate and build synergies between diverse constraint technologies, and the success they enjoyed suggest a necessity of a principled and general study of methods to develop such hybrid solvers. [16] provides a study of the relationship between various CASP solvers highlighting the importance of creating unifying approaches to describe such systems. For instance, the CASP systems ACSOLVER, CLINGCON, and EZCSP came to being within two consecutive years. These systems rely on different ASP and CSP technologies, so it is difficult to clearly articulate their similarities and differences. In addition, the CASP solvers adopt different communication schemas among their heterogeneous solving components. The system EZCSP adopts a "*black-box*" architecture, whereas ACSOLVER and CLINGCON advocate tighter integration. The crucial message transpiring from these developments in the CASP community is the ever growing need for standardized techniques to integrate computational methods spanning multiple research areas. Currently such an integration requires nontrivial expertise in multiple areas, for instance, in SAT, ASP and CSP. We argue for undertaking an effort to mitigate difficulties of designing hybrid reasoning systems by identifying general principles for their development and studying the implications of various design choices.

As a step in this direction, in this paper we conduct a case study aiming to explore a crucial aspect in building hybrid systems – the integration schemas of participating solving methods. We study various integration schemas and their performance, using CASP as our test-bed domain. As an exemplary subject for our study we take the CASP system EZCSP. Originally, EZCSP was developed as an inference engine for CASP that allowed a lightweight, *black-box*, integration of ASP and CSP. In order for our analysis to be conclusive we found it important to study the different integration mechanisms using the same technology. Within the course of this work we implemented "*grey-box*" and "*clear-box*" approaches for combining ASP and CSP reasoning within EZCSP. We evaluate these configurations of EZCSP on three domains – Weighted Sequence, Incremental Scheduling, and Reverse Folding – from the *Model and Solve* track of the *Third Answer Set Programming Competition – 2011* (ASPCOMP) [1]. Hybrid paradigms such as CASP allow for mixing language constructs and computational mechanisms stemming from different formalisms. Yet, one may design encodings that favor only single reasoning capabilities of a hybrid system. For this reason, in our study we evaluate different encodings for the proposed benchmarks that we call "pure ASP", "true CASP", and "pure CSP". As a result we expect to draw a comprehensive picture comparing and contrasting various integration schemas on several kinds of encodings possible within hybrid approaches.

We start with a brief review of the CASP formalism. Then we draw a parallel to SMT solving, aimed at showing that it is possible to transfer to SMT the results obtained in this work for CASP solving. In Section 3 we review the integration schemas used in the design of hybrid solvers focusing on the schemas implemented in EZCSP within this project. Section 4 provides a brief introduction to the application domains considered, and discusses the variants of the encodings we compared. Experimental results and their analysis form Section 5.

2 Review: the CASP and SMT problems

The review of logic programs with constraint atoms follows the lines of [15]. A *regular program* is a finite set of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_l, not\ a_{l+1}, \ldots, not\ a_m,\ not\ not\ a_{m+1}, \ldots, not\ not\ a_n, \qquad (1)$$

where a_0 is $\bot$ or an atom, and each a_i ($1 \leq i \leq n$) is an atom. This is a special case of programs with nested expressions [18]. We refer the reader to [18] for details on the definition of an answer set of a logic program. A *choice rule* construct $\{a\}$ [20] of the LPARSE language can be seen as an abbreviation for a rule $a \leftarrow\ not\ not\ a$ [9]. We adopt this abbreviation.

A constraint satisfaction problem (CSP) is defined as a triple $\langle X, D, C\rangle$, where X is a set of variables, D is a domain of values, and C is a set of constraints. Every constraint is a pair $\langle t, R\rangle$, where t is an n-tuple of variables and R is an n-ary relation on D. An *evaluation* of the variables is a function from the set of variables to the domain of values, $\nu : X \to D$. An evaluation ν *satisfies* a constraint $\langle (x_1, \ldots, x_n), R\rangle$ if $(v(x_1), \ldots, v(x_n)) \in R$. A *solution* is an evaluation that satisfies all constraints.

Consider an alphabet consisting of regular and constraint atoms, denoted by $\mathcal{A}$ and $\mathcal{C}$ respectively. By $\tilde{\mathcal{C}}$, we denote the set of all literals over $\mathcal{C}$. The constraint literals are identified with constraints via a function $\gamma : \tilde{\mathcal{C}} \to C$ so that for any literal l, $\gamma(l)$ has a solution if and only if $\gamma(\bar{l})$ does not have one (where $\bar{l}$ denotes a complement of l). For a set Y of constraint literals over $\mathcal{C}$, by $\gamma(Y)$ we denote a set of corresponding constraints, i.e., $\{\gamma(c) \mid c \in Y\}$. Furthermore, each variable in $\gamma(Y)$ is associated with a domain. For a set M of literals, by M^+ and $M^{\mathcal{C}}$ we denote the set of positive literals in M and the set of constraint literals over $\mathcal{C}$ in M, respectively.

A *logic program with constraint atoms* is a regular logic program over an extended alphabet $\mathcal{A} \cup \mathcal{C}$ such that, in rules of the form (1), a_0 is $\bot$ or $a_0 \in \mathcal{A}$. Given a logic program with constraint atoms Π, by $\Pi^{\mathcal{C}}$ we denote Π extended with choice rules $\{c\}$ for each constraint atom c occurring in Π. We say that a consistent and complete set M of literals over atoms of Π is an *answer set* of Π if

(a1) M^+ is an answer set of $\Pi^{\mathcal{C}}$ and
(a2) $M^{\mathcal{C}}$ has a solution.

The CASP problem is the problem of determining, given a logic program with constraint atoms Π, whether Π has an answer set.

For example, let Π be the program

$$\begin{array}{l} am \leftarrow\ X < 12 \\ lightOn \leftarrow\ switch, not\ am \\ \{switch\} \\ \bot \leftarrow not\ lightOn. \end{array} \tag{2}$$

Intuitively, this program states that (a) *light* is *on* only if an action of *switch* occurs during the *pm* hours and (b) *light* is *on* (according to the last rule in the program). Consider a domain of X to be integers from 0 till 24. It is easy to see that a set

$$\{switch, lightOn, \neg\ am, \neg X < 12\}$$

forms the only answer set of program (2).

One may now draw a parallel to satisfiability modulo theories (SMT). To do so we first formally define the SMT problem. A *theory* T is a set of closed first-order formulas. A CNF formula F (a set of clauses) is T*-satisfiable* if $F \wedge T$ is satisfiable in the first-order sense. Otherwise, it is called T-unsatisfiable. Let M be a set of literals. We sometimes may identify M with a conjunction consisting of all its elements. We say that M is a T-model of F if

(m1) M is a model of F and
(m2) M is T*-satisfiable*.

The SMT problem for a theory T is the problem of determining, given a formula F, whether F has a T-model. It is easy to see that in the CASP problem, Π^C in condition (a1) plays the role of F in (m1) in the SMT problem. At the same time, the condition (a2) is similar in nature to the condition (m2).

Given this tight conceptual relation between the SMT and CASP formalisms, it is not surprising that solvers stemming from these different research areas share a lot in common in their design even though these areas have been developing to a large degree independently (CASP being a much younger field). We start next section by reviewing major design principles/methods in crafting SMT solvers. We then discuss how CASP solvers follow one or another method. This discussion allows us to systematize solvers' design patterns present both in SMT and CASP so that their relation becomes clearer. Such transparent view on solvers' architectures immediately translates findings in one area into another. Thus although the case study conducted in this research uses CASP technology only, we expect similar results to hold for SMT, and for the construction of hybrid automated reasoning methods in general.

3 SMT/CASP Integration Schemas

Satisfiability modulo theories (SMT) integrates different theories "under one roof". Often it also integrates different computational procedures for processing such hybrid theories. We are interested in these synergic procedures explored by the SMT community over the past decade. We follow [21, Section 3.2] for a review of several integration techniques exploited in SMT.

In every discussed approach, a formula F is treated as a satisfiability formula where each of its atoms is considered as a propositional symbol, *forgetting* about the theory T. Such view naturally invites an idea of *lazy* integration: the formula F is given to a SAT solver, if the solver determines that F is unsatisfiable then F is T-unsatisfiable as well. Otherwise, a propositional model M of F found by the SAT solver is checked by a specialized T-solver which determines whether M is T-satisfiable. If so, then it is also a T-model of F, otherwise M is used to build a clause C that precludes this assignment, i.e., $M \not\models C$ while $F \cup C$ is T-satisfiable if and only if F is T-satisfiable. The SAT solver is invoked on an augmented formula $F \cup C$. Such process is repeated until the procedure finds a T-model or returns unsatisfiable. Note how in this approach two automated reasoning systems – a SAT solver and a specialized T-solver – interleave: a SAT solver generates "candidate models" whereas a T-solver tests whether these models are in accordance with requirements specified by theory T. We find that it is convenient to introduce the following terminology for the future discussion: a *base* solver and a *theory* solver, where a base solver is responsible for generating candidate models and *theory* solver is responsible for any additional testing required for stating whether a candidate model is indeed a solution.

It is easy to see how the lazy integration policy translates into the realm of CASP. Given a program with constraint atoms Π, an answer set solver serves the role of a base solver by generating answer sets of $\Pi^{\mathcal{C}}$ (that are "candidate answer sets" for Π) and then uses a CLP/CSP solver as a theory solver to verify whether condition (a2) is satisfied on these candidate answer sets. Constraint answer set solver EZCSP embraces the lazy integration approach in its design.[3] To solve the CASP problem, EZCSP offers a user several options for *base* and *theory* solvers. For instance, it allows for the use of answer set solvers CLASP [10], CMODELS [12], DLV [5] as base solvers and CLP systems SICSTUS PROLOG [24] and BPROLOG [26] as theory solvers. Such variety in possible configurations of EZCSP illustrates how lazy integration provides great flexibility in choosing underlying base and theory solving technology in addressing problems of interest.

The Davis-Putnam-Logemann-Loveland (DPLL) procedure [6] is a backtracking-based search algorithm for deciding the satisfiability of a propositional CNF formula. DPLL-like procedures form the basis for most modern SAT solvers as well as answer set solvers. If a DPLL-like procedure underlies a base solver in the SMT and CASP tasks then it opens a door to several refinements of lazy integration. We now describe these refinements that will also be a focus of the present case study.

In the lazy integration approach a base solver is invoked iteratively. Consider the SMT task: a CNF formula F_{i+1} of the $i+1^{\text{th}}$ iteration to a SAT solver consists of a CNF formula F_i of the i^{th} iteration and an additional clause (or a set of clauses). Modern DPLL-like solvers commonly implement such technique as *incremental* solving. For instance, incremental SAT-solving allows the user to solve several SAT problems $F_1, \ldots, F_n$ one after another (using single invocation of the solver), if F_{i+1} results from F_i by adding clauses. In turn, the solution to F_{i+1} may benefit from the knowledge obtained during solving $F_1, \ldots, F_i$. Various modern SAT-solvers, including MIN-

[3] [2] refers to lazy integration of EZCSP as *lightweight* integration of ASP and constraint programming.

ISAT [7,8], implement interfaces for incremental SAT solving. Similarly, the answer set solver CMODELS implements an interface that allows the user to solve several ASP problems $\Pi_1, \ldots, \Pi_n$ one after another, if Π_{i+1} results from Π_i by adding a set of rules whose heads are $\bot$. It is natural to utilize incremental DPLL-like procedures for enhancing the lazy integration protocol: we call this refinement *lazy+* integration. In this approach rather than invoking a base solver from scratch an incremental interface provided by a solver is used to implement the iterative process.

[21] also reviews such integration techniques used in SMT as *on-line SAT solver* and *theory propagation*. In the on-line SAT solver approach, the T-satisfiability of the (partial) assignment is checked incrementally, while the assignment is being built by the DPLL-like procedure. This can be done fully eagerly as soon as a change in the partial assignment occurs or at some regular intervals, for instance. Once the inconsistency is detected, a SAT solver is instructed to backtrack. The theory propagation approach extends the on-line SAT solver technique by allowing a theory solver not only to verify that a current partial assignment is T-consistent but also to detect literals in a CNF formula that must hold given the current partial assignment. The CASP solver CLINGCON exemplifies the implementation of the theory propagation integration schema in CASP by unifying answer set solver CLASP as a base solver and constraint processing system GECODE. ACSOLVER and IDP systems are other CASP solvers that implement the theory propagation integration schema.

Three Kinds of EZCSP**:** To conduct our analysis of various integration schemas and their effect on the performance of the hybrid systems we used the CASP solver EZCSP as a baseline technology. As mentioned earlier, original EZCSP implements the lazy integration schema. In the course of this work we developed enhanced interfaces with answer set solver CMODELS that allowed for the two other integration schemas: lazy+ integration and on-line SAT solver. These implementations rely on API interfaces provided by CMODELS allowing for varying level of integration between the solvers. The development of these API interfaces in CMODELS was greatly facilitated by the API interface provided by MINISAT v. 1.12b supporting non-clausal constraints [8]. In the following we call

- EZCSP implementing lazy integration with CMODELS as a base solver – a *black-box*.
- EZCSP implementing lazy+ integration with CMODELS – a *grey-box*.
- EZCSP implementing on-line SAT solver integration with CMODELS (fully eagerly) – a *clear-box*.

In all these configurations of EZCSP we assume BPROLOG to serve in the role of a theory solver.

4 Application Domains

In this work we compare and contrast different integration schemas of hybrid solvers on three application domains that stem from various subareas of computer science. This section provides a brief overview of these applications. All benchmark domains are from the *Third Answer Set Programming Competition – 2011* (ASPCOMP) [1], in

particular, the *Model and Solve* track. We chose these domains for our investigation as they display features that benefit from the synergy of computational methods in ASP and CSP. Each considered problem contains variables ranging over a large integer domain thus making grounding required in pure ASP a bottleneck. On the other hand, the modeling capabilities of ASP and availability of such sophisticated solving technique as learning makes ASP attractive for designing solutions to these domains. As a result, CASP languages and solvers become a natural choice for these benchmarks making them ideal for our investigation.

Three Kinds of CASP Encodings: It is easy to note that hybrid languages such as CASP allow for mix-and-match constructs and processing techniques stemming from different formalisms. Yet, any pure ASP encoding of a problem is also a CASP formalization of the same problem. Similarly, it is possible to encode a problem in such a way that only the CSP solving capabilities of the CASP paradigm are employed. In this study for two of the benchmarks we considered three kinds of encodings in the CASP language of EZCSP: *pure-ASP* encoding; *pure-CSP* encoding; and *true-CASP* encoding. In the third benchmark, the use of three distinct encodings was not possible because both ASP and CSP features play a major role in the efficiency of the computation.

Analysis of these varying kinds of encodings in CASP gives us a better perspective on how different integration schemas are effected by the design choices made during the encoding of a problem. At the same time considering the encoding variety allows us to verify our intuition that true-CASP is an appropriate modeling and solving choice for the explored domains.

The **weighted-sequence** (WSEQ) domain is a handcrafted benchmark problem. Its key features are inspired by the important industrial problem of finding an optimal join order by cost-based query optimizers in database systems. [17] provides a complete description of the problem itself as well as the formalization that became "golden standard" in this work, i.e., the formalization named SEQ++.

In the weighted-sequence problem we are given a set of leaves (nodes) and an integer m – maximum cost. Each leaf is a pair *(weight, cardinality)* where *weight* and *cardinality* are integers. Every sequence (permutation) of leaves is such that all leaves but the first are assigned a *color* that, in turn, associates a leaf with a *cost* (via a cost formula). A colored sequence is associated with the *cost* that is a sum of leaves' costs. The task is to find a colored sequence with cost at most m. We refer the reader to [17] for the details of pure-ASP encoding SEQ++. The same paper also contains the details on a true-CASP variant of SEQ++ in the language of CLINGCON. We further adapted that encoding to the language of EZCSP by means of simple syntactic transformations. Here we provide a review of details of the SEQ++ formalization that we find most relevant to this presentation. The non-domain predicates of the pure-ASP encoding are $leafPos$, $posColor$, $posCost$. Intuitively, $leafPos$ is responsible for assigning a position to a leaf, $posColor$ is responsible for assigning a color to each position, $posCost$ carries information on costs associated with each leaf. The main difference between the pure-ASP and true-CASP encodings is in the treatment of the cost values of the leaves. We first note that cost predicate $posCost$ in the pure-ASP encoding is "functional". In other words, when this predicate occurs in an answer set its first argument uniquely determines its second argument. Often, such functional predicates in ASP en-

codings can be replaced by constraint atoms in CASP encodings. Indeed, this is the case in the weighted-sequence problem. Thus in the true-CASP encoding, predicate $posCost$ is replaced by constraint atoms, making it possible to evaluate cost values by CSP techniques. This approach is expected to benefit performance especially when the cost values are large. Predicates $leafPos$ and $posColor$ are also functional. The pure-CSP encoding is obtained from the true-CASP encoding by replacing $leafPos$ and $posColor$ predicates by constraint atoms.

The **incremental scheduling** (IS) domain stems from a problem occurring in commercial printing. In this domain, a schedule is maintained up-to-date with respect to jobs being added and equipment going offline. A problem description includes a set of devices, each with predefined number of instances (slots for jobs), and a set of jobs to be produced. The penalty for a job being tardy is computed as $td \cdot imp$, where td is the job's tardiness and imp is a positive integer denoting the job's importance. The total penalty of a schedule is the sum of the penalties of the jobs. The task is to find a schedule whose total penalty is no larger than the value specified in a problem instance. We direct the reader to [3] for a complete description of the domain. The pure-CSP encoding used in our experiments is the official competition encoding submitted to ASPCOMP by the EZCSP team. In this encoding, constraint atoms are used for (i) assigning start times to jobs, (ii) selecting which device instance will perform a job, and (iii) calculating tardiness and penalties. The true-CASP encoding was obtained from the pure-CSP encoding by introducing a new relation $on\_instance(j, i)$, stating that job j runs on device-instance i. This relation and ASP constructs of the EZCSP language replaced the constraint atoms responsible for the assignment of device instances in the pure-CSP encoding. The pure-ASP encoding was obtained from the true-CASP encoding by introducing suitable new relations, such as $start(j, s)$ and $penalty(j, p)$, to replace all the remaining constraint atoms.

In the **reverse folding** (RF) domain, one manipulates a sequence of n pairwise connected segments located on a 2D plane in order to take the sequence from an initial configuration to a goal configuration. The sequence is manipulated by pivot moves: rotations of a segment around its starting point by 90 degree in either direction. A pivot move on a segment causes the segments that follow to rotate around the same center. Concurrent pivot moves are prohibited. At the end of each move, the segments in the sequence must not intersect. A problem instance specifies the number of segments, the goal configuration, and required number of moves, t. The task is to find a sequence of exactly t pivot moves that produces the goal configuration. The true-CASP encoding used for our experiments is from the official ASPCOMP 2011 submission package of the EZCSP team. In this encoding, relation $pivot(s, i, d)$ states that at step s the i^{th} segment is rotated in direction d. The effects of pivot moves are described by constraint atoms, which allow carrying out the corresponding calculations with CSP techniques. The pure-ASP encoding was obtained from the true-CASP encoding by adopting an ASP-based formalization of the effects of pivot moves. This was accomplished by introducing two new relations, $tfoldx(s, i, x)$ and $tfoldy(s, i, y)$, stating that the new start of segment i at step s is $\langle x, y \rangle$. The definition of the relations is provided by suitable ASP rules.

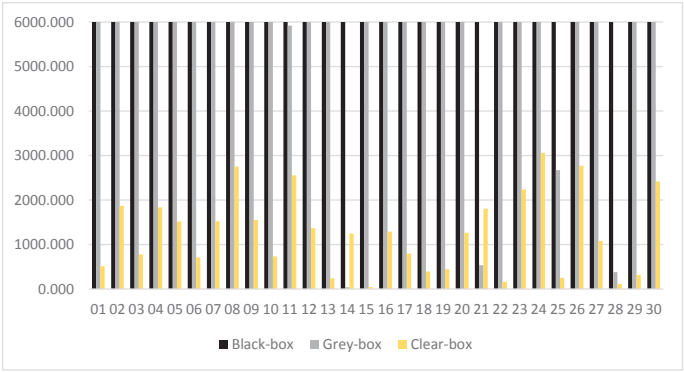

Figure 1. Performance on WSEQ domain: true-CASP encoding

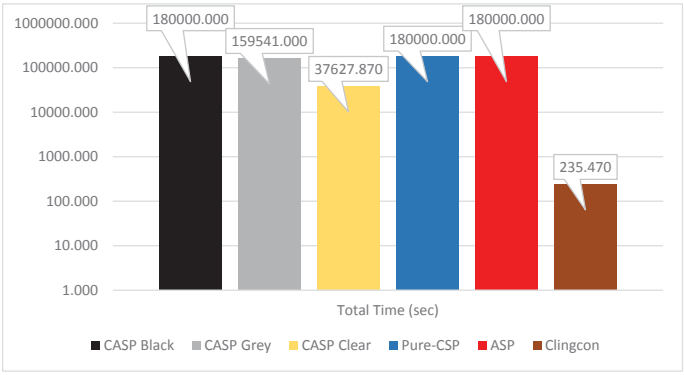

Figure 2. Performance on WSEQ domain: total times in logarithmic scale

5 Experimental Results

The experimental comparison of the integration schemas was conducted on a computer with an Intel Core i7 processor running at 3GHz. Memory limit for each process and timeout considered were 3 GB RAM and $6,000$ seconds respectively. The version of EZCSP used in the experiments was 1.6.20b49: it incorporated CMODELS version 3.83 as a base solver and BPROLOG 7.4_3 as a theory solver. Answer set solver CMODELS 3.83 was also used for the experiments with the pure-ASP encodings. In order to provide a frame of reference with respect to the state of the art in CASP, the tables for WSEQ and IS include performance information for CLINGCON 2.0.3 on true-CASP encodings adapted to the language of CLINGCON. The ezcsp executable used in the experiments and the encodings can be downloaded from *http://www.mbalduccini.tk/ezcsp/aspocp2013/ezcsp-binaries.tgz* and

http://www.mbalduccini.tk/ezcsp/aspocp2013/experiments.tgz respectively. In all figures presented:

- CASP Black, CASP Grey, CASP Clear denote EZCSP implementing respectively *black-box*, *grey-box* and *clear-box*, and running a true-CASP encoding;
- Pure-CSP denotes EZCSP implementing *black-box* running a pure-CSP encoding (note that for pure-CSP encodings there is no difference in performance between the integration schemas);
- ASP denotes CMODELS running a pure-ASP encoding;
- Clingcon denotes CLINGCON running a true-CASP encoding.

We begin our analysis with WSEQ. The instances used in the experiments are the 30 instances available via ASPCOMP. WSEQ proves to be a domain that truly requires the interaction of the ASP and CSP solvers. Answer set solver CMODELS on the pure-ASP encoding runs out of memory on every instance (in the tables, out-of-memory conditions and timeouts are both rendered as out-of-time results). EZCSP on the pure-CSP encoding reaches the timeout limit on every instance. The true-CASP encoding running in *black-box* also times out on every instance. As shown in Figure 1, the true-CASP encoding running in *grey-box* performs slightly better. The true-CASP encoding running in *clear-box* instead performs *substantially* better. Figure 2 reports the total times across all the instances for all solvers/encodings pairs considered. Notably, CASP solver CLINGCON on true-CASP encoding is several orders of magnitude faster than any other configuration. This confirms that for this domain tight integration schemas indeed have an advantage. Recall that CLINGCON implements a tighter integration schema than that of EZCSP *clear-box* that, in addition to the on-line SAT solver schema of *clear-box*, also includes theory propagation. Answer set solver CLASP serves the role of base solver of CLINGCON whereas GECODE is the theory solver.

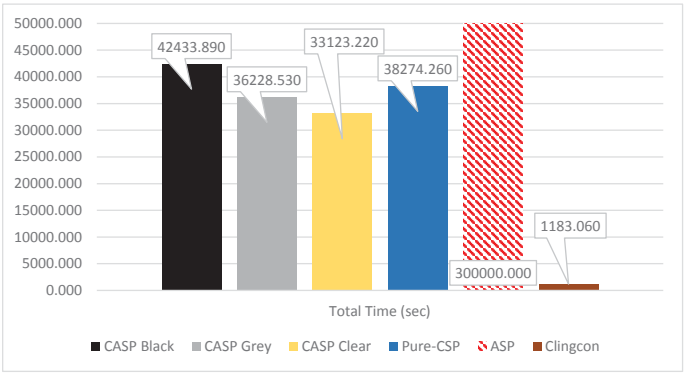

Figure 3. Performance on IS domain, easy instances: total times (ASP encoding off-chart)

In case of the IS domain we considered two sets of experiments. In the former we used the 50 official instances from ASPCOMP. We refer to these instances as *easy*.

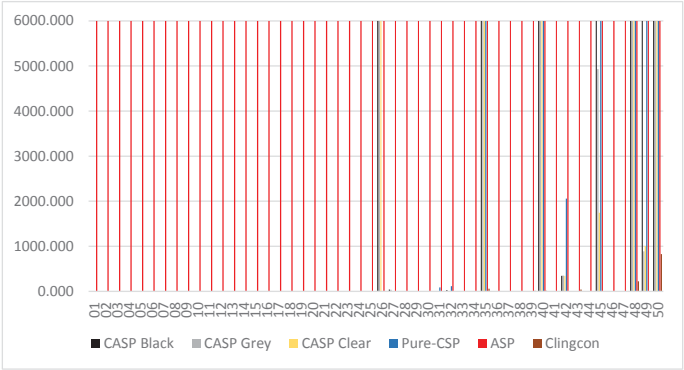

Figure 4. Performance on IS domain, easy instances: overall view

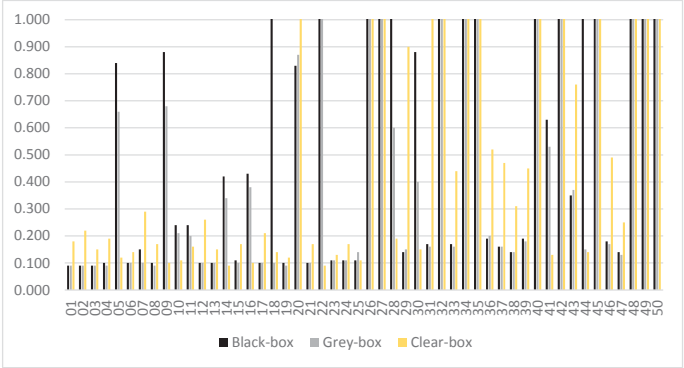

Figure 5. Performance on IS domain, easy instances: true-CASP encoding (detail of 0-1sec execution time range)

Figure 4 depicts the overall per-instance performance on the IS-easy domain. It appears that tight integration schemas have an advantage, allowing the true-CASP encoding to outperform the pure-CSP encoding. As one might expect, the best performance for the true-CASP encoding is obtained with the *clear-box* integration schema, as shown in Figure 3 and in Figure 5. Figure 3 provides a comparison of the total times. In this case the early pruning of the search space made possible by the *clear-box* architecture yields substantial benefits. As expected, it is also the case that *grey-box* is faster than *black-box*. As for WSEQ, CLINGCON is the fastest, and CMODELS on the pure-ASP encoding runs out of memory in all the instances.

The next experiment reveals an interesting change in behavior of solver/encodings pairs as the complexity of the instances of the IS domain grows. In this test, we used a set of 30 instances obtained by (1) generating randomly 500 fresh instances; (2) running the true-CASP encoding with the *grey-box* integration schema on them with a timeout

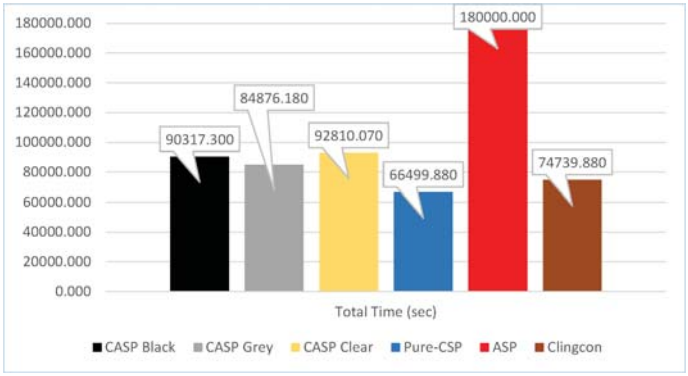

Figure 6. Performance on IS domain, hard instances: overall view

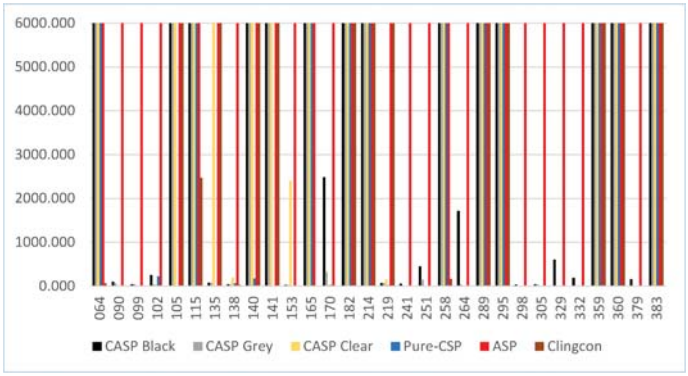

Figure 7. Performance on IS domain, hard instances: total times

of 300 seconds; (3) selecting randomly, from those, 15 instances that resulted in time-out and 15 instances that were solved in 25 seconds or more. The numerical parameters used in the process were selected with the purpose of identifying challenging instances. The overall per-instance execution times reported in Figure 7 clearly indicate the level of difficulty of the selected instances. Remarkably, these more difficult instances are solved more efficiently by the pure-CSP encoding that relies only on the CSP solver, as evidenced by Figure 6. In fact, the pure-CSP encoding outperforms every other method of computation, *including* CLINGCON on true-CASP encoding. More specifically, solving the instances with the true-CASP encoding takes between 30% and 50% longer than with the pure-CSP encoding. (Once again, CMODELS runs out of memory.)

The final experiment focuses on the RF domain. We used the 50 official instances from ASPCOMP to conduct the analysis. Figure 9 presented shows that this domain is comparatively easy. Figure 10 illustrates that the *black-box* and *grey-box* integration schemas are several orders of magnitude faster than *clear-box*. This somewhat surpris-

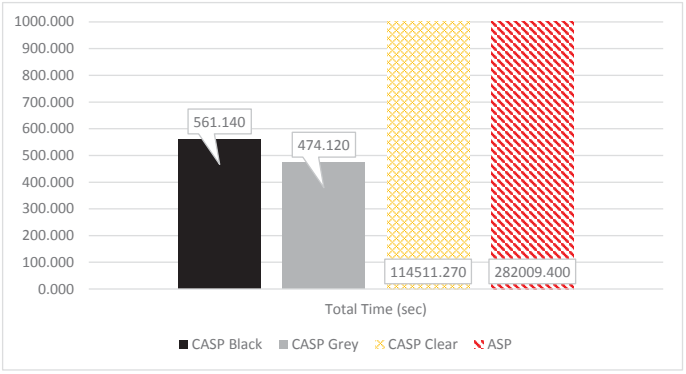

Figure 8. Performance on RF domain: total times (detail of 0-1000sec execution time range, ASP and *clear-box* off-chart)

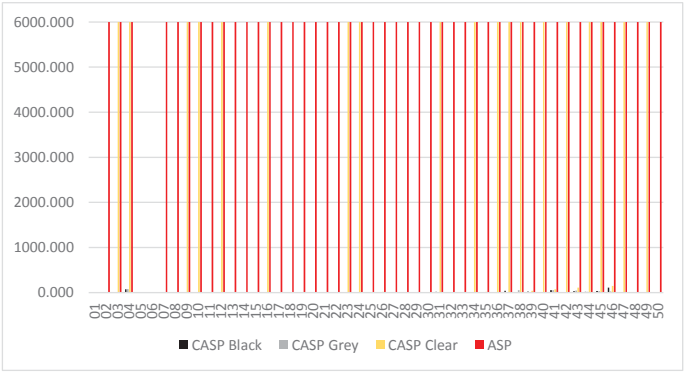

Figure 9. Performance on RF domain: overall view

ing result can be explained by the fact that in this domain frequent checks with the theory solver add more overhead rather than being of help in identifying an earlier point to backtrack. CMODELS on the pure-ASP encoding runs out of memory in all but 3 instances. The total execution times are presented in Figure 8.

6 Conclusions

The case study conducted in this work clearly illustrates the influence that integration methods have on the behavior of hybrid systems. Each integration schema may be of use and importance for some domain. Thus systematic means ought to be found for facilitating building hybrid systems supporting various coupling mechanisms. Building clear and flexible API interfaces allowing for various types of interactions between the solvers seems a necessary step towards making the development of hybrid solving

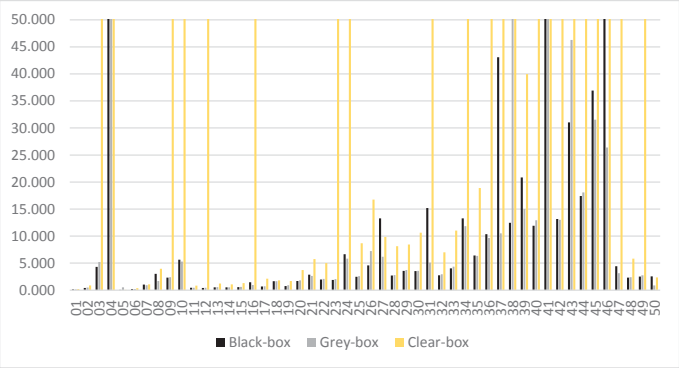

Figure 10. Performance on RF domain: true-CASP encoding, detail of 0-0.50sec execution time range

systems effective. This work provides evidence for the need of an effort to this ultimate goal.

References

1. Third answer set programming competition (2011), https://www.mat.unical.it/aspcomp2011/
2. Balduccini, M.: Representing constraint satisfaction problems in answer set programming. In: Proceedings of ICLP'09 Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'09) (2009)
3. Balduccini, M.: Industrial-Size Scheduling with ASP+CP. In: Delgrande, J.P., Faber, W. (eds.) 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR11). Lecture Notes in Artificial Intelligence (LNCS), vol. 6645, pp. 284–296. Springer Verlag, Berlin (2011)
4. Brewka, G., Niemelä, I., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011)
5. Citrigno, S., Eiter, T., Faber, W., Gottlob, G., Koch, C., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: The DLV system: Model generator and application frontends. In: Proceedings of Workshop on Logic Programming (WLP97) (1997)
6. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM 5(7), 394–397 (1962)
7. Een, N., Biere, A.: Effective preprocessing in sat through variable and clause elimination. In: SAT (2005)
8. Een, N., Sörensson, N.: An extensible sat-solver. In: SAT (2003)
9. Ferraris, P., Lifschitz, V.: Weight constraints as nested expressions. Theory and Practice of Logic Programming 5, 45–74 (2005)
10. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI'07). pp. 386–392. MIT Press (2007)
11. Gebser, M., Ostrowski, M., Schaub, T.: Constraint answer set solving. In: Proceedings of 25th International Conference on Logic Programming (ICLP). pp. 235–249. Springer (2009)

12. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. Journal of Automated Reasoning 36, 345–377 (2006)
13. Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, pp. 89–134. Elsevier (2008)
14. Jaffar, J., Maher, M.: Constraint logic programming: A survey. Journal of Logic Programming 19(20), 503–581 (1994)
15. Lierler, Y.: Constraint answer set programming (2012), http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127221
16. Lierler, Y.: On the relation of constraint answer set programming languages and algorithms. In: Proceedings of the 26th Conference on Artificial Intelligence (AAAI-12). MIT Press (2012)
17. Lierler, Y., Smith, S., Truszczynski, M., Westlund, A.: Weighted-sequence problem: Asp vs casp and declarative vs problem oriented solving. In: Fourteenth International Symposium on Practical Aspects of Declarative Languages (2012), http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127085
18. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence 25, 369–389 (1999)
19. Mellarkod, V.S., Gelfond, M., Zhang, Y.: Integrating answer set programming and constraint logic programming. Annals of Mathematics and Artificial Intelligence (2008)
20. Niemelä, I., Simons, P.: Extending the Smodels system with cardinality and weight constraints. In: Minker, J. (ed.) Logic-Based Artificial Intelligence, pp. 491–521. Kluwer (2000)
21. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM 53(6), 937–977 (2006)
22. Rossi, F., van Beck, P., Walsh, T.: Constraint porgramming. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, pp. 181–212. Elsevier (2008)
23. Schuller, P., Patoglu, V., Erdem, E.: A Systematic Analysis of Levels of Integration between Low-Level Reasoning and Task Planning. In: Workshop on Combining Task and Motion Planning at the IEEE International Conference on Robotics and Automation 2013 (2013)
24. SICStus: Sicstus Prolog Web Site (2008), http://www.sics.se/isl/sicstuswww/site/
25. Wittocx, J., Mariën, M., Denecker, M.: The IDP system: a model expansion system for an extension of classical logic. In: LaSh. pp. 153–165 (2008)
26. Zhou, N.F.: The language features and architecture of B-Prolog. Journal of Theory and Practice of Logic Programming (TPLP) 12(1–2), 189–218 (Jan 2012)